Implementation of a Testbed Authentication Architecture for Mobile IP

Report

Edward Okoko
Department of Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand

November, 2005

Abstract

Achieving Internet access "anywhere, anytime" is an ideal that much research has been working towards. One aspect of this research is the integration o different network technologies such as wireless local area network (WLAN) technologies and third generation (3G) technologies into so-called heterogeneous networks. With such integration, an important issue is handling mobility both within a given network technology and among the various ones. Mobile IP is a standard that supports transparent mobility at the network layer. However, another issue that arises is authentication and authorisation of users. The Mobile IP standard does not adequately address this issue. Proposals for improved authentication architectures for Mobile IP have therefore been made. This document describes the implementation of an authentication architecture prototype that extends the functionality of an existing Mobile IP implementation.

Contents

Chapter 1

Introduction

Mobile computing is expected to increase in popularity. Indeed, currently, the usage and spread of wireless local area networks (WLANs) is on the increase. The IEEE 802.11 standard (commonly referred to as WiFi) is the dominant standard for WLANs. WLAN hotspots continue to be deployed in locations such as cafes, airports, libraries, and so on. Mobile computing hinges on wireless access to the Internet. The desire among mobile computing users is the provision of such access "anywhere, anytime". However, WLANs cannot, by definition, satisfy this need on their own. Because of the limited coverage they offer, something more is required.

3G technologies such as CDMA2000 and UMTS, on the other hand, provide wide coverage. But they, on their own, are not a panacea either. 3G networks typically provide significantly lower data rates than WLANs. For this reason, an active area of research is heterogeneous networks. (Such networks are also referred to as integrated or 4G networks.) Heterogeneous networks consist of more than one access technology. This allows for the provision of the best of both worlds. For instance, a 3G technology could be combined with a WLAN technology. In such a situation, if a mobile station is within range of a WLAN and the signal is sufficiently strong, then that is the preferred technology as higher data rates are supported. When the WLAN signal strength falls below a threshold, then the mobile station switches to the 3G network.[3]

There are a variety of issues to be resolved in integrating networks. With the illustration given, an immediately evident issue is that of handoff. A mobile station needs to be able to determine when to switch from one network technology to another. Such handoffs are referred to as vertical handoffs. (Horizontal handoffs on the other hand, involve moving between cells within the same network technology. In a WLAN, for instance, this would involve moving from within the range of one access point to the range of another access point.) In addition, it is undesirable to have the station continually switching from one network technology to the next. This could happen, for example, when the WLAN signal strength fluctuates close to the threshold. Without some sort of control, the station would flap constantly between the WLAN and the 3G network. It is desirable then, to have some sort of hysteresis implemented so that switching between network technologies is controlled.

The issue of handoff described above is one of mobility management. Another question in mobility management regards the layer at which network mobility should be provided in the context of heterogeneous networks. Mobility can be provided at various layers of the TCP/IP reference model such as the network, transport and application layers. For instance, Mobile IP, the mobility protocol relevant to this project, is a network layer solution to mobility. Determining which layer is best suited to providing mobility has been debated with differing viewpoints given. Banerjee et al [2] suggest the application layer as being the best choice while Eddy [10] suggests the tranport layer as "the strongest candidate". Any choice has tradeoffs associated with it because mobility is a "feature with no well defined place in classical protocol stacks." [10]

A few of the other issues to be addressed in heterogeneous networks are quality of service, profile handling, and AAA support. A more detailed discussion of the issues surrounding heterogeneous networks can be found in [12] and [15].

The aim of this project is to extend Dynamics HUT Mobile IP, an open-source implementation of the Mobile IPv4 protocol. This implementation was carried out at the Helsinki University of Technology. The purpose of the extension is to provide the AAA functionality that is lacking in the current implementation. The motivation for this is that such functionality would be available in open-source. The Mobile IP protocol, AAA infrastructure and the combination of Mobile IP and AAA will be discussed in the "Background" chapter.

The rest of the document is structured as follows. The following chapter presents the background for this project. Specifically, it outlines Mobile IP, AAA infrastructure, and the rationale for combining Mobile IP and AAA. Related work is also presented. The subsequent chapter describes in detail, the work carried out in this project. The design of the extensions proposed and the extent of the implementation are presented. The chapter that then follows presents the results of the implementation. These results are presented in the form of output from the implementation to show how the implementation actually works. The final chapter gives an appraisal of the extent of the implementation in this project. In addition, it offers suggestions on further work that could be done with regard to the project.

Chapter 2

Background

2.1 Mobile IPv4

Mobile IP is a network layer solution to the problem of mobility. It essentially solves the problem of mobility via the use of two IP addresses: a home address and a care-of address. TCP, the most commonly used protocol at the transport layer, makes use of a 4-tuple for its connections:

<source IP address, source port, destination IP address, destination port>

If any one of these changes, an existing connection will be broken. The rationale behind the operation of Mobile IP is preventing this from happening. In particular, it ensures that the IP addresses do not change.

2.1.1 Mobile IPv4 Operation

Mobile IPv4 consists of three functional entities:

- the mobile node
- the home agent
- the foreign agent

These are shown in Figure 2.1.

A mobile node possesses a static home address and a dynamic care-of address. The home address is associated with the mobile node's home network. The home address is used to give the illusion of the mobile node being in its home network. The care-of address, on the other hand, changes with the mobile node's point of attachment. The care-of address allows the mobile node to continue to receive packets addressed to it even when it is not connected to its home network. A care-of address may either be a foreign agent one or a colocated one. A foreign agent care-of address is an IP address of a foreign agent. A colocated care-of address, on the other hand, is an IP address temporarily assigned to one of the interfaces of the mobile node.

The operation of Mobile IP can be divided into three mechanisms [24], namely:

- Discovering the care-of address
- Registering the care-of address
- Tunnelling to the care-of address

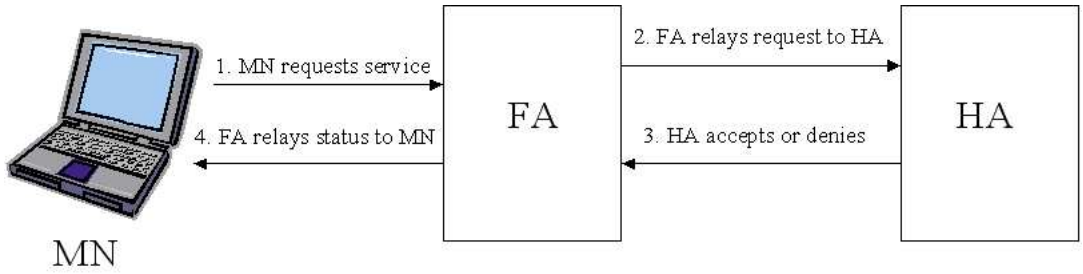

Figure 2.1: Mobile IP registration process. FA is the foreign agent, HA is the home agent, and MN is the mobile node.

Discovery: When a mobile node attaches itself to a foreign network, it needs to discover a care-of address. The mobile node typically determines the care-of address from agent advertisements, which contain information about care-of addresses. Agent advertisements are an extension to standard router advertisements. Foreign agents (and home agents) typically broadcast agent advertisements at regular intervals. A mobile node may, however, broadcast a solicitation that will be answered by any foreign agent or home agent that receives it.

Registration: Once a mobile node has a care-of address, it informs its home agent of the fact. This is necessary for the home agent to determine where to route packets addressed to the mobile node. This process is called registering to the home agent. The mobile node sends (usually via the foreign agent) a registration request to its home agent. If the home agent approves the request, it updates itself with the information provided in the request and sends a reply to the mobile node. The sequence of Mobile IP registration operations is shown in Figure 2.1.

Tunnelling: The home agent redirects packets from the home network to the care-of address provided by the mobile node. This redirection is done via tunnelling. Tunnelling is done by encapsulating a packet addressed to the home network within a new header addressed to the care-of address. The packet can thus be routed to the location of the mobile node in a foreign network. Once at the foreign network, the packet is then stripped of the new header so that it appears to have the mobile node's home address as the destination IP address. It is in this original form that the packet is presented to higher level protocols such as TCP. In the case of a foreign agent care-of address, a tunnel from a home agent terminates at the foreign agent. However, with a colocated care-of address, the tunnel terminates at an interface on the mobile node.

2.2 AAA

AAA stands for Authentication, Authorisation and Accounting. The AAA model provides a framework for these three aspects of access control. The need for the AAA architecture is borne out of a need for scalability. Without AAA, managing the wide variety of network equipment and the associated authentication methods would be difficult. The following questions mimic the operation of the framework [13]:

- Who are you?
- What services am I allowed to give you?
- What did you do with my services while you were using them?

Authentication

Authentication is the process of verifying the credentials of an entity. This requires that a given entity possesses unique credentials that allow for unambiguous identification. Examples of such credentials include the combination of a user ID and password, digital certificates, and so on. Authentication is carried out prior to granting access to resources.

Authorisation

Authorisation is the process of determining the resources a given entity is permitted to access and granting access to these resources. Authentication and authorisation are typically carried out simultaneously in an environment managed by AAA.

Accounting

Accounting refers to the methodology for collecting information on resource usage. This could be necessary for a variety of reasons including billing, auditing, trend analysis, capacity-planning, and so on.

2.2.1 AAA Applied

The AAA model is application-neutral. As a result implementations of AAA architecture can vary in complexity as dictated by specific needs. The model is based on a client-server model. An AAA server maintains a database of authentication and authorisation information on users. An AAA client, on the other hand, resides on a network component such as a network access server.[19] The AAA server and client communicate to provide users with distributed AAA services.

A good example of where AAA infrastructure could be useful is the case of Internet Service Providers (ISPs). An ISP needs to be able to determine the identity of users of its network resources. It also has to be able to determine what resources a given user is allowed to access. For instance, one user on a "low-cost" plan may get limited bandwidth while another user on a "higher" plan may be offered more bandwidth. Very closely tied to this is collecting information on resource usage. Using the example of the two users, beyond just knowing how much bandwidth has been allocated to each, it may also be important to know such things as the duration of the connection or the amount of data transferred. These details may then be neccessary for billing purposes. For instance, a user who exceeds the allocated data transfer limit may need to be charged extra. This example requires the existence of some sort of policy. AAA provides a unified framework for the implementation of such a policy.

RADIUS (Remote Access Dial-In User Service) [30] is the most widely used AAA protocol.[19] It was actually designed prior to the notion of AAA as a framework. Despite this, it satisfies most of the requirements of AAA. RADIUS was originally designed to support dialup connections. Its current usage, though, is more varied. RADIUS is a client-server protocol. A user connects to a NAS – on which a RADIUS client resides – seeking network access. The RADIUS client forwards the user's identification data to a RADIUS server. The RADIUS server then replies to the NAS accepting or rejecting the user's connection request. In the case of acceptance, the RADIUS server informs the NAS of the services that a user is authorised to receive. Communication between a RADIUS client and a RADIUS server is protected by means of a shared secret used to authenticate messages exchanged between the entities. RADIUS uses UDP for its protocol packets. RADIUS does have shortcomings as an AAA protocol, particularly in the wake of mobility. These shortcomings include the limitation on the size of attribute values, the lack of well-defined fault tolerance mechanisms, and the lack of end-to-end security.

The Diameter protocol [5] has been developed to address the shortcomings of RADIUS. Its development reflects the growth in the number of Internet users, access technologies, and network

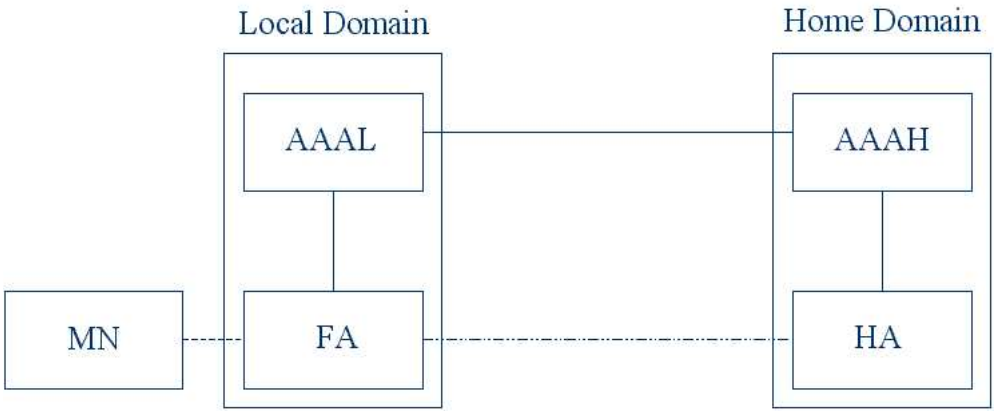

Figure 2.2: The basic model

access servers (which, in turn, have also grown in complexity). In addition, there is a need for the ability to handle network access across administrative domains as well as roaming. These changes in the Internet place new demands on AAA protocols. [5] Diameter, unlike RADIUS, is a peer-to-peer protocol. The improvements in Diameter over RADIUS include better fault tolerance, better security features, reliable transport (via the use of TCP or the Stream Control Transmission Protocol (SCTP) rather than UDP), server-initiated message support (useful, for instance, if a server would like to request the termination of a connection), and support for roaming.

2.3 Integrating Mobile IP and AAA

The design of Mobile IP "allowed for, but did not [include], verification protocols to check the mobile node's identity in the absence of a preconfigured security association."[25] In other words, the standard operation of Mobile IP assumes security associations between the mobile node and foreign agent (MN-FA), the foreign agent and home agent (FA-HA), and the mobile node and home agent (MN-FA). However, in the interests of facilitating connectivity anywhere, it is reasonable to imagine a mobile node attempting to gain access in an administrative domain other than that of its home network. RFC 2977 [11] describes requirements for integrating Mobile IP and AAA functionality. This project is based on the requirements outlined in the RFC.

The basic model is shown in Figure 2.2. When a client (the mobile node) is in a foreign domain, it typically needs to request resources provided by the domain. To do this, the client presents its credentials to an attendant (which in Mobile IP would be the foreign agent). The attendant, however, may not have enough information to verify the client's credentials. The attendant therefore forwards the credentials on to the local authority (AAAL). The local authority too may have insufficient information to verify the client's credentials. It is, however, able to contact an external authority to verify the client's credentials. This external authority is the home authority (AAAH) of the client. The local authority's role is therefore to first obtain authorisation from an external authority and then notify the attendant of this. Once the attendant is informed that a given client is authorised, it grants the client access to services that it is entitled to.

The requirements outlined above implicitly assume a security model. First, there needs to be a security association between the mobile node and home authority. This represents a significant difference from the original definition of Mobile IP, which requires a security association between the mobile node and the home agent. The home agent is, however, a routing agent rather than an authentication agent. Secondly, it is assumed that there is a security association between the

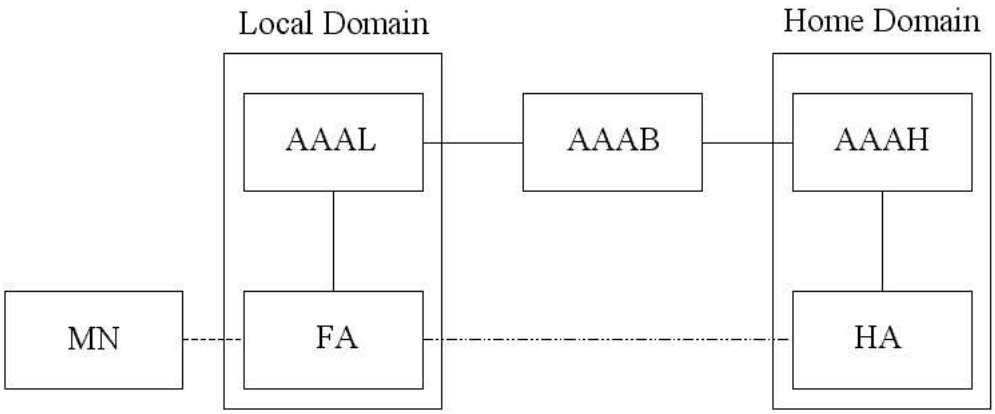

Figure 2.3: The extended model with a broker.

attendant (foreign agent) and the local authority. This is necessary because the foreign agent needs to be able to trust the local authority's role in verifying credentials. Thirdly, there needs to be a security association between the local authority and the home authority. Finally, there needs to be a security association between the home authority and the home agent. It should be noted that in this security model there is only one security association maintained by the mobile node. This is desirable to limit the amount of configuring necessary within the mobile node.

The means to identify a mobile node suggested by the RFC is the Network Access Identifer [4], an identifier of the form *user@realm*. This is to be used for mobile nodes that do not already have an IP address. The advantage of using this form of an identifier is that the home domain, *realm*, can be easily determined by a local authority.

One problem with the AAA model as described here is that of scalability. With this arrangement, every local authority (and, by extension, foreign domain) would need to establish security associations with every home authority that a mobile node could possibly have a security association with. To solve this problem, RFC 2977 suggests using a brokered model (see Figure 2.3) as an extension to the basic model described here. A broker (AAAB), in this model, would establish security associations with as many administrative domains as possible. This way, a local authority would ideally need to have a security association with only one broker. The broker, on the other hand, acts as an intermediary for passing AAA data between local and home authorities.

Wireless network operators need to be able to charge for the services they offer. The original design of Mobile IP was geared towards developing an enabling technology for transparent mobility. The notion of billing was not part of this design. AAA offers this notion. In essence, with the combination of the Mobile IP and AAA, a business case for providing Internet access using Mobile IP can be made.

2.4 Related work

2.4.1 Previous Research

There are several proposals in the literature on authentication and mobility. Wang et al [37] describe an "authentication architecture for fast authentication during inter-networking handoff and large-scale heterogeneous networks." Cappiello et al [6] also make proposals in a similar vein. Yang et al [38] suggest simultaneously using a public-key cryptosystem and a symmetric key cryptosystem to carry out secure Mobile IP registration.

Table 2.1: Mobile IP implementations

Name	IP Version	Operating System	License
Monarch	4	FreeBSD	BSD Style
Secure Mobile Net	4	FreeBSD	BSD Style
Dynamics	4	Linux	GPLv2
Secgo Mobile IP	4	Linux/Windows	Commercial
MosquitoNet	4	Linux	GPL?
Birdstep	4	Windows	Commercial
Ecutel	4	Windows	Commercial
SHISA	6	FreeBSD	BSD Style
INRIA HMIPv6	6	FreeBSD	BSD Style
Lancaster MIPv6	6	Linux	?
MIPL	6	Linux	GPL

What appears to be lacking, however, is implementations that are compliant with RFC 2977. This project aims to lead towards addressing this.

2.4.2 Mobile IP Implementations

There are several implementations of Mobile IP both commercial and free. There are implementations available for both Mobile IPv4 and Mobile IPv6. A brief comparison of some of the features in the various implementations is presented in Table 2.1.

Most of the available free implementations of Mobile IPv4 in particular have not been actively updated recently. This can probably be explained by the fact that there are several commercial implementations of Mobile IPv4 available, which suggests maturity among Mobile IPv4 implementations. Among the free Mobile IPv4 implementations available for Linux, Dynamics is the most complete one. It has been shown to be a stable version and appears to be favoured amongst researchers. However, it has not been updated since September 2001.

Chapter 3

Project Implementation

This chapter describes the design and implementation of the project. Firstly, an overview that motivates the specific aims of the project is given. The next section then provides the design of the implementation followed by a section that outlines the steps of the implementation. Subsequent sections detail these implementation steps.

3.1 Overview

As discussed in the background chapter, RFC 2977 [11] details the requirements for integrating Mobile IP and AAA. According to the RFC, the only security association a mobile node is required to maintain is between itself and the home authority (that is, the home AAA server). Using this security association, it is possible to derive a security association between the mobile node and its home agent, as well as another between the mobile node and a foreign agent as required for the standard operation of Mobile IP. RFC 2977 does not, however, specify how this derivation can be done. This role is fulfilled by RFC 3957 [23], which describes AAA registration keys for Mobile IPv4. More specifically, RFC 3957 details "extensions to Mobile IP registration messages that can be used to create Mobility Security Associations between the mobile node and its home agent, and/or between the mobile node and a foreign agent."

It is useful to further distinguish the two RFCs referred to in the previous paragraph. RFC 2977 is concerned with describing interactions between Mobile IP and AAA. Through this interaction, it becomes possible for AAA servers to authenticate and authorise network access requests from mobile nodes. In Mobile IP, registration requests are used to request network access. To comply with RFC 2977, Mobile IP agents need extra functionality to translate between the Mobile IP registration messages and AAA messages. RFC 3957, therefore, describes the format of extensions to Mobile IP registration messages to incorporate the information required for interacting with AAA servers.

In terms of compliance to RFCs, extending the functionality of Mobile IP entities to handle registration messages with the extensions specified in RFC 3957 may be considered a necessary condition for fulfilling the requirements of RFC 2977. Implementing such functionality is to a large extent the focus of this project. The aim of the project was to extend a freely available implementation of Mobile IP, such that it would be able to properly handle registration requests with AAA extensions and make use of the keys generated from these requests. An additional requirement, to comply with RFC 2977, is carrying out all needed AAA and Mobile IP functions for an initial registration message within a single Internet traversal.

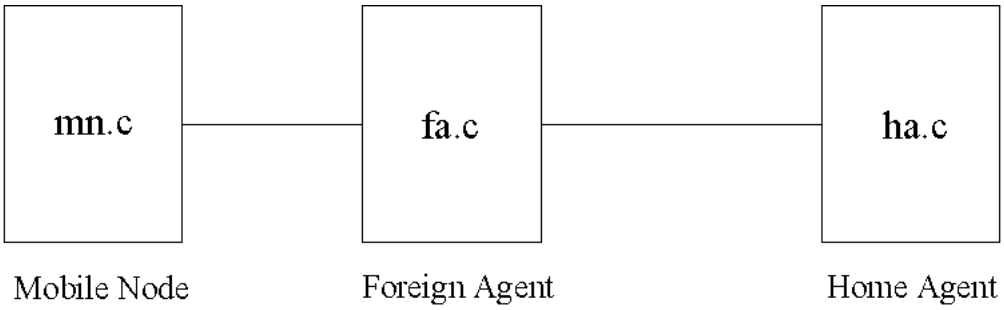

Figure 3.1: Simplified representation of Dynamics implementation.

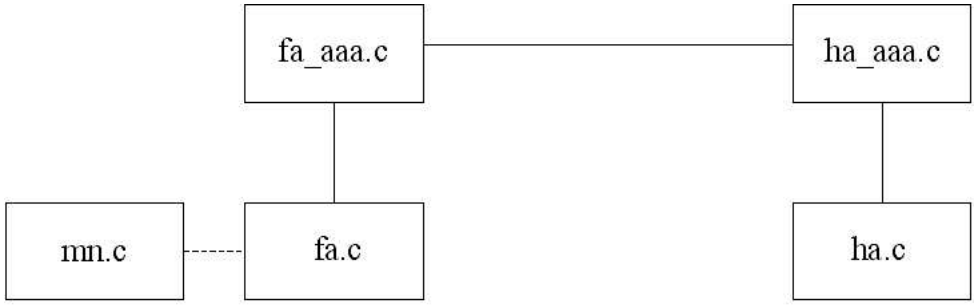

Figure 3.2: Design of the project implementation

3.2 Implementation Design

The design of Dynamics Mobile IP is modelled on the functional entities defined by the Mobile IP standard. From an operational perspective, the starting points of Dynamics implementation are the source files, `mn.c` (for the mobile node), `fa.c` (for the foreign agent) and `ha.c` (for the home agent). This setup is shown in Figure 3.1. These files are referred to here as "starting points" because there are several other files associated with each of them. In the case of the foreign agent, for instance, functions to handle requests are defined in the source file `fa_request.c`. Therefore, the functionality in these three files basically coordinates the behaviour of the given entity.

The design of the project implementation is modelled directly on the basic model of the authentication architecture described in RFC 2977 (see Figure 2.2). The design for this project is shown in Figure 3.2. As can be seen in the figure, the source files `fa_aaa.c` and `ha_aaa.c` are intended to provide the functionality for the local authority and home authority, respectively, of the combined Mobile IP and AAA authentication architecture. The project implementation has been designed to be as cleanly separated as possible from the existing Dynamics implementation. The primary advantage of this is limiting the changes to the existing implementation. On the other hand, the main disadvantage of the decision is duplication of some of the functionality provided in Dynamics. In a fully functional implementation, `fa_aaa.c` and `ha_aaa.c` would interact with AAA servers. However, this project was intended to be a proof-of-concept. Therefore, interaction with AAA servers is considered to be outside the project's scope.

3.3 Implementation Outline

Choosing C as the programming language for the implementation, provided the most straightforward approach to integrating the extension implemented in this project with the original implementation of Dynamics. The Dynamics software was implemented in C. The steps carried out in the implementation were as follows:

- Walking through the code to become familiar with the Dynamics HUT Mobile IP implementation.
- Setting up an initial testbed to run the software. This testbed would then form the basis for running the functionality implemented in this project.
- Building an independent AAA prototype to test and implement the AAA functionality independently of the Dynamics implementation.
- Finally, incorporating the functionality of the AAA prototype built in the previous step into the Dynamics implementation.

These steps are described in detail in the sections that follow.

3.4 Code Walkthrough

The first step towards implementation was becoming familiar with the source code. This essentially involved walking through the code. Despite the absence of a document that describes the structure of the Dynamics implementation, the code is well structured with descriptive comments for most functions. The initial focus of the walkthroughs was the operational aspects of the foreign agent (`fa.c` and associated files) and home agent (`ha.c` and associated files) particularly with regard to handling registration requests. The reason for this was to determine where to intercept a request using AAA extensions that would otherwise be rejected by the foreign agent and home agent programs. The walkthroughs were useful in getting a feel for the design of the Dynamics implementation. However, using these alone it proved challenging to establish exactly how the programs worked.

3.5 Initial Testbed Setup

The next step therefore, was setting up a testbed to run the Dynamics software in debug mode using the following command:

```
<daemon> --fg --debug
```

where `<daemon>` is `dynfad` (for the foreign agent), `dynhad` (for the home agent) or `dynmnd` (for the mobile node) as required. In debug mode, the programs give detailed output on the state of the program. Often, it is possible to determine which function has produced a given line of output because the function name is given as well. However, even when this is not the case, it is a simple matter to search the source code to find the function that produced the output. This proved very useful for the overall implementation process.

It was also necessary to determine how much AAA functionality had been implemented. According to the documentation that covers the Dynamics implementation's conformance to RFCs, AAA functionality had been fully implemented in the mobile node and only partially implemented in the foreign agent and home agent. Carrying out this exercise involved manipulating the configuration file of the mobile node to see how both the foreign agent and home agent handled various

configurations.

The AAA functionality of the mobile node needed to be tested too. Usefully, a program to do this is included in the Dynamics package. The program, `aaasimul`, acts as a foreign agent and simulates the functionality of a home agent and the AAA infrastructure. The Dynamics package contains functions in `auth.c` to handle AAA key generation. When `aaasimul` receives a request from a mobile node, it makes use of these routines to generate the MN-HA and MN-FA keys. The program then sends a simulated registration accepted reply to the mobile node. As a result, `aaasimul` demonstrates what is involved in generating both keys via AAA extensions and the replies. When this is combined with the requests with AAA extensions from the mobile node, one obtains a complete (albeit simulated) picture of the process involved. The source file for `aaasimul`, `aaasimul.c`, was used as the basis for the implementation in the project.

3.6 Building an AAA Prototype

The actual implementation of the project begun by focusing attention on `aaasimul.c`. The aim here was twofold:

- Firstly, to separate the role of foreign agent and local authority from that of the home agent and the home authority.
- Secondly, to enable the home agent to actually create a tunnel to the mobile node rather than simply sending a reply simulating the creation of the tunnel (as is done with `aaasimul`).

Doing this facilitated building an AAA prototype independently of the Dynamics implementation. The prototype so built would, in turn, form the basis for extending the functionality of Dynamics. The source files written for the purpose of building the prototype are `aaa_fa.c` and `aaa_ha.c` for the foreign domain and home domain, respectively.

`aaasimul.c` consists of two functions, `send_agent_adv()` and `reply_msg()`, besides `main()`. `send_agent_adv()` broadcasts agent advertisements to notify any mobile nodes of the foreign agent's presence. This function was placed in `aaa_fa.c` to maintain the same purpose. The extra functionality provided in `aaa_fa.c` is acting as a router between the mobile node and the home agent. Essentially, this entails forwarding all packets from the mobile node to the home agent and vice versa. The code to do this is shown below:

```
/* forward message to aaa_ha */
if ((udp_sock2 = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
  return -1;
}

to.sin_family = AF_INET;
to.sin_port = htons(434);
inet_aton("192.168.242.1", &(to.sin_addr));
memset(&(to.sin_zero), '\0', 8);

res = sendto(udp_sock2, buf, len, 0,
             (struct sockaddr *) &to, tolen);

replylen = recvfrom(udp_sock2, buf2, BUFSIZE, 0,
                    (struct sockaddr *) &to, &tolen);

sendto(udp_sock, buf2, replylen, 0,
       (struct sockaddr *) &from, fromlen);
```

In the implementation of `aaasimul.c`, `udp_sock` and `from` define the file descriptor and socket structure, respectively, to the mobile node. In addition to these, as shown in the code, `aaa_fa.c` defines `udp_sock2` and `to` as the file descriptor and socket structure, respectively, to the home agent.

The other function in `aaasimul.c`, `reply_msg()` was placed in `aaa_ha.c`. In `aaasimul.c`, the purpose of `reply_msg()` is to generate keys and send a reply to the mobile node simulating request acceptance. In `aaa_ha.c` this is extended by actually creating a tunnel between the home agent and the mobile node. Creating such a tunnel requires the use of a colocated care-of address. The definition of the function for the creation of a tunnel in `aaa_ha.c`, `create_tunnel()`, is given below:

```
void create_tunnel(struct bindingentry *binding, struct hashtable *hash)
{
        struct in_addr temp_addr, local;
        int ok = 1;

        memset((char *) &local, 0, sizeof(local));
        inet_aton((char *) "192.168.242.1", &local);

        memset((char *) &temp_addr, 0, sizeof(temp_addr));
        inet_aton((char *) "192.168.240.2", &temp_addr);

        if(tunnel_fetch(hash, temp_addr, TUNNEL_IPIP, 0) == NULL){
                inet_aton((char *) "192.168.242.2", &binding[0].mn_addr);
                inet_aton((char *) "192.168.240.2", &binding[0].lower_addr);
                if(ok && (tunnel_add(hash, binding[0].lower_addr, binding[0].tun_dev,
                                local, 1, TUNNEL_IPIP, 0) == NULL)){
                        printf("tunnel_add failed \n");
                        ok = 0;
                }

                if(ok && dyn_ip_route_replace(binding[0].mn_addr, binding[0].tun_dev) != 0){
                  ok = 0;
                }
        }
}
```

The function first checks whether there is an existing tunnel for the given `binding` and `hash`. This is done via the call to `tunnel_fetch()`, a function defined in `tunnel.c`, which is part of the Dynamics package. If there is an existing tunnel, no further action is taken. If there is no tunnel, one is created with the call to `tunnel_add()` which is also defined in `tunnel.c`. If the tunnel is successfully created, `dyn_ip_route_replace()` is called to update the host's routing table with the route to the mobile node via the tunnel.

To summarise, a prototype was built to which a mobile node could make a request for registration using AAA extensions. With this prototype, a successful reply would be sent to the registration. In addition, a tunnel to the mobile node would be set up.

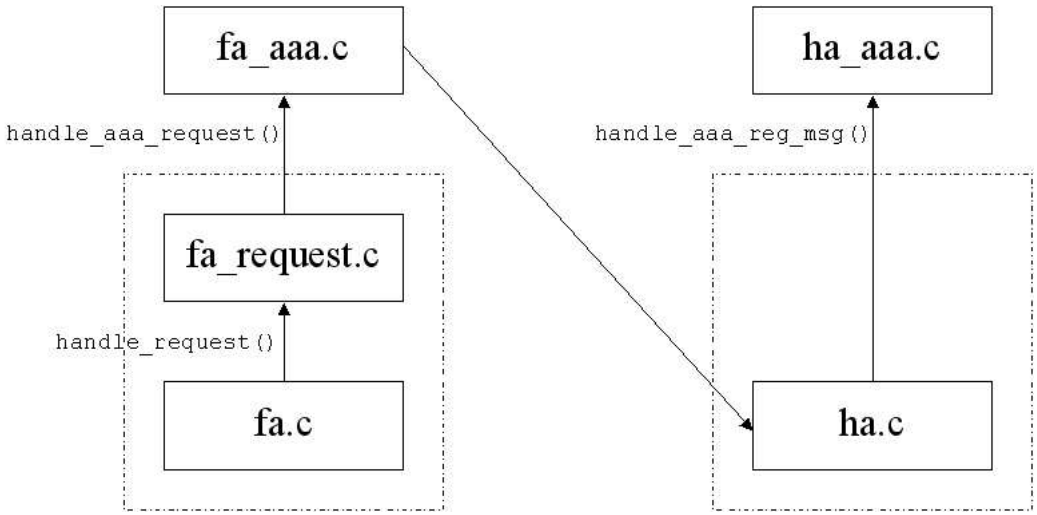

Figure 3.3: The interactions within the project implementation.

3.7 Integrating the Prototype with Dynamics Mobile IP

Having completed building the AAA prototype with `aaa_fa.c` and `aaa_ha.c`, the next step was to incorporate the functionality provided into the foreign agent and home agent modules.

The interactions within the project's implementation are shown in Figure 3.3. The figure depicts the manner in which a registration request with AAA extensions is intended to be handled. The registration request is first received by a function in `fa.c`. This function calls `handle_request()`, a function defined in `fa_request.c`. If the request is determined to contain AAA extensions, it is then to be handled within `fa_aaa.c`. This is done by calling `handle_aaa_request()`, a function defined in `fa_aaa.c`. The request is then forwarded to the home agent. In a similar fashion, when the home agent determines the function contains AAA extensions, it is forwarded for handling within `ha_aaa.c`. This is done by calling `handle_aaa_reg_msg()`, a function defined in `ha_aaa.c`. It is worth noting here that the interaction between the functionality in `fa_aaa.c` and `ha_aaa.c`, rather than being direct, goes through `ha.c`. For the project implementation, this is necessary because it is at `ha.c` that registration messages are received.

The source files `fa.c`, `fa_request.c` and `fa.c` are part of the existing Dynamics implementation. The dotted boxes in the diagram are intended to portray a boundary around the existing implementation. The source files `fa_aaa.c` and `ha_aaa.c`, on the other hand, were defined for this project. The design of the project is, therefore, that for simple Mobile IP requests – that is, those that do not require the use of AAA functionality – Dynamics should work as originally implemented. The implementation of this project is then intended to provide the additional AAA functionality on top of the existing implementation when such functionality is required. The details of the implementation are presented in the following sections.

3.8 Modifications to Foreign Agent

It has been noted that a significant difference between RFC 2977, which details AAA requirements for Mobile IP, and RFC 2002, the original specification of Mobile IP, is the requirement that a mobile node have a security association with the home authority rather than the home agent. In the Dynamics implementation therefore, when validating a request from a mobile node, the first

check carried out the foreign agent is whether a mobile-home authentication extension is included. This is done with the following code in the function `validate_request()` found in `fa_request.c`:

```
if (ext->mh_auth == NULL) {
        error_text = "no mh_auth";
        error_code = REGREP_BAD_REQUEST_FA;
}
```

If the mobile-home authentication extension is missing, a reply is sent to the mobile node informing it that its request has been rejected. The error code `REGREP_BAD_REQUEST_FA` in the Dynamics implementation corresponds to the reply code of 70, which is defined in the Mobile IP standard as a registration rejection by a foreign agent due to a "poorly formed Request". In the project implementation therefore, this situation was handled by checking the registration request for the presence of AAA extensions prior to the calling of the `validate_request()`. The function `handle_request()` (also defined in `fa_request.c`) calls `validate_request()` as follows:

```
if (validate_request(ext, info))
        return 1;
```

For the purpose of handling requests with AAA extensions that `validate_request()` cannot handle, the code that follows was placed before the `if` statement above:

```
if(ext->mh_auth == NULL && ext->mn_aaa_auth != NULL){
        if(handle_aaa_request(ext, info)){
                return 0;
        }else{
                return 1;
        }
}
```

The purpose of the code in the snippet above is to check for the case of a registration request that has AAA authentication extensions but not a mobile-home agent authentication extension. Registration requests that satisfy these criteria are passed to the function `handle_aaa_request()`, which is described shortly. The call to this function effectively bypasses the rest of the functionality in `handle_request()`.

3.8.1 handle_aaa_request()

The function `handle_aaa_request()` is defined in `fa_aaa.c` one of the project source files. Its role is to forward all registration requests with AAA requests to the home authority. The function first copies the registration request and the authentication extensions as shown in the following code snippet:

```
/* copy the MAC protected area directly from the received message */
auth_ext = (char *) ext->mn_aaa_auth;
auth_len = GET_GEN_AUTH_EXT_LEN(ext->mn_aaa_auth);

...

len = (__u8 *) auth_ext - (__u8 *) ext->req + auth_len;
if (len < sizeof(struct reg_req) + auth_len || len > MAXMSG) {
        return 1;
}
memcpy(msg, ext->req, len);
```

The difference between `auth_ext` and `ext->req` in the calculation of `len` above gives the length of registration request alone. `auth_len` stores the length of the MN-AAA authentication extensions as calculated by `GET_GEN_AUTH_EXT_LEN`, a macro defined in `message.h`. The variable `len` therefore stores the total length of the registration request and the authentication extensions. This value is used to copy the request and the authentication extensions into `msg` as shown above in the call to `memcpy()`.

The registration request is then checked to determine if it contains a challenge with the following code:

```
if (ext->challenge != NULL && (char *) ext->challenge >= copy_end) {
        int n = GET_CHALLENGE_EXT_LEN(ext->challenge);
        if (len + n > MAXMSG)
                return 1;
        memcpy(msg + len, ext->challenge, n);
        len += n;
        DEBUG(DEBUG_FLAG, " * adding Challenge ext (len=%i)\n", n);
}
```

The variable `copy_end` in the `if` statement is used simply to check that the challenge comes after the authentication extension as required by the standard that specifies it. The macro `GET_CHALLENGE_EXT_LEN`, also defined in `message.h`, is used to determine the length of the challenge. The essential point of all this is to add the challenge extension to `msg` as well.

The steps followed so far in the function with regard to copying the registration message into `msg` are drawn directly from `aaasimul.c`. The extra functionality in `fa_aaa.c` is relaying the request to the home authority and relaying the reply from the home agent to the mobile node. This is done with the following code:

```
to_aaah.sin_port = htons(FA_DEFAULT_UDP_PORT);
inet_aton("192.168.242.1", &(to_aaah.sin_addr));
...
sent = sendto(udp_sock_aaah, msg, len, 0,
                     (struct sockaddr *) &to_aaah, to_aaah_len);
...
replylen = recvfrom(udp_sock_aaah, buf, BUFSIZE, 0,
                    (struct sockaddr *) &to_aaah, &to_aaah_len);

sent = sendto(info->iface->udp_sock, buf, replylen, 0,
               (struct sockaddr *) &info->src, sizeof(info->src));
```

Essentially, the request stored in `msg` is forwarded to the home authority at the IP address contained in `to_aaah` via the socket `udp_sock_aaah`. The reply is received and stored in `buf`. This reply is forwarded to the mobile node at the IP address in `info->src` via the socket `info->iface->udp_sock`.

3.9 Modifications to Home Agent

The Dynamics implementation of the home agent, like that of the foreign agent, validates all registration requests from a mobile node. Once again, the home agent first verifies the presence of the mobile-home authentication extensions. This is done with a function also named `validate_request()` defined in `ha.c`. The following code shows how this validation is carried out:

```
if (ext->req == NULL || ext->mh_auth == NULL) {
        LOG2(LOG_WARNING,
```

```
                "FA %s: message does not contain either request "
                "and/or message authentication\n", faaddrstr);
        if (ext->mh_auth == NULL)
                *code = REGREP_MN_FAILED_AUTH_HA;
        else
                *code = REGREP_BAD_REQUEST_HA;
        return NULL;
}
```

In this case, the presence of both a request and the authentication extensions is checked. If the registration request informaton is missing, a reply indicating registration denial by the home agent on the basis of a "poorly formed request" (reply code 134 in RFC 2002) is sent to the mobile node, shown as `REGREP_BAD_REQUEST_HA` in the code. More relevantly, if the authentication extension is missing, a registration denial with a reply code of 131 for "mobile node failed authentication" (shown as `REGREP_MN_FAILED_AUTH_HA`) is sent to the mobile node.

The home agent validates a registration request with the following function call within `handle_reg_msg()` defined in `ha.c`:

```
mn_spi = validate_request(binding, cli_addr.sin_addr, msg, n, &ext,
                          &code, &auth_type);
```

For this project, the following code was added just before the function call above:

```
if((ext.mn_aaa_auth != NULL) && (ext.mh_auth == NULL)){
        handle_aaa_reg_msg(&cli_addr, &ext, binding, bindings, tunnels, &code,
                           &bindingcount, t_data, config.max_bindings, rsock);
        return 0;
}
```

As was done in the foreign agent, the message is first checked to determine if it has MN-AAA authentication extensions and does not have a mobile-home authentication extension. Messages that do not satisfy these criteria are handled as usual by Dynamics. However, those messages that satisfy the criteria are handled by `handle_aaa_reg_msg()`, which is defined in `ha_aaa.c`.

3.9.1 handle_aaa_reg_msg()

The function `handle_aaa_reg_msg()`, like `reply_msg()` in `aaa_ha.c`, is responsible for generating keys, creating a tunnel to the mobile node and sending a registration reply to the mobile node. As part of its integration with the Dynamics implementation, it also adds an entry for the mobile node to the table of bindings maintained by the home agent.

The following code snippet demonstrates the generation of the MN-HA key within `handle_aaa_reg_msg()`:

```
mn_spi.shared_secret_len = MAXSHAREDSECRETLEN;
printf("auth_generate_key: alg=%i, secret=aaa-test, "
       "keymat_len=%i, nodeaddrlen=%i\n",
       MN_HA_KEYGEN_ALG, KEYMAT_LEN, nodeaddrlen);
if (auth_generate_key(
                      MN_HA_KEYGEN_ALG, MN_AAA_SECRET, MN_AAA_SECRET_LEN,
                      (unsigned char *) (keymat + 1), KEYMAT_LEN,
                      nodeaddr, nodeaddrlen,
                      mn_spi.shared_secret, &mn_spi.shared_secret_len)) {
  printf("MN-HA key generation failed\n");
  return;
```

The significant point about the code shown is the use of the variable `mn_spi`. This variable is important because it is will be needed in the process of creating a binding for the mobile node with the home agent, as will be detailed later. The code above shows that MN-HA key generated is stored in `mn_spi.shared_secret`.

A binding is then created by calling `create_binding()` as shown below:

```
binding = create_binding(ext, &mn_spi, bindings, tunnels, bcount,
                         max_bindings, 1);
```

If successful, the call creates a tunnel to the mobile node and a binding entry based on the information passed to it in `ext` and `mn_spi`. It returns a pointer to the binding entry created. The function `create_binding` is discussed in greater detail later.

Bindings have a specified lifetime. This information is added to the binding entry within `handle_request()` with the following code:

```
binding->mod_time = time(NULL);
binding->timeout = MIN(ntohs(ext->req->lifetime), mn_spi.max_lifetime);
binding->exp_time = time(NULL) + binding->timeout;
```

The time of the most recent modification to the binding, `mod_time`, is set to the current time. (The creation time of a binding is also recorded in a binding entry. This, however, is set in `create_binding()`.) `timeout` is set to the lesser of the lifetime specified in the registration request and the maximum lifetime specified in `mn_spi`. The time of the expiry of the binding, `exp_time`, is therefore set to be that of `timeout` added to the current time. If a binding entry is added to the binding table without these values being set, the entry (along with the associated tunnel) will be deleted when the table is subsequently checked for expired bindings.

```
tunnel_data = (struct ha_tunnel_data *) binding->data;
tunnel_data->reverse_tunnel = (ext->req->opts & REGREQ_REVERSE_TUNNEL) != 0;
tunnel_data->encapsulation = ENCAPS_IPIP;
```

The information on the source of a registration request is stored as shown:

```
tunnel_data->lower_saddr.s_addr = cli_addr->sin_addr.s_addr;
binding->lower_port = cli_addr->sin_port;
```

`tunnel_data->lower_saddr.s_addr` contains the IP address from which the registration was received while `binding->lower_port` contains the port number.

Once the binding entry has all the information required, it is added to the binding table as follows:

```
binding_add(bindings, binding);
```

The function `binding_add()` is defined in `bindings.c`, one of the Dynamics source files. In the function call, `bindings` is the binding table to which the entry `binding` is added.

The final step is to send a reply informing the mobile node that its registration request has been accepted and that it has successfully been connected to the mobile node:

```
dest_addr.sin_family = AF_INET;
dest_addr.sin_addr.s_addr = tunnel_data->lower_saddr.s_addr;
dest_addr.sin_port = binding->lower_port;

res = sendto(sock, buf, pos - buf, 0, (struct sockaddr *) &dest_addr,
             sizeof(dest_addr));
```

3.9.2 create_binding()

The function `create_binding()` is used to create a binding entry and a tunnel to the mobile node.

Before a binding entry is created, the function checks whether the maximum number of bindings for the home agent has been reached. This is done with the following code:

```
if (*bcount >= max_bindings) {
        printf("max bindings reached\n");
        return NULL;
}
```

`b_count` points to the number of bindings in the home agent's binding table. If the maximum number of bindings, `max_bindings` has been reached, no more bindings are created. On the other hand, if the maximum number of bindings has not been reached then memory is allocated for the binding as shown:

```
binding = malloc(sizeof(struct bindingentry));
...
memset(binding, 0, sizeof(struct bindingentry));
binding->data = malloc(sizeof(struct ha_tunnel_data));
...
memset(binding->data, 0, sizeof(struct ha_tunnel_data));
t_data = (struct ha_tunnel_data *) binding->data;
```

The `data` element of the `bindingentry` structure is a pointer to an unspecified type (`void *`). For the purposes of the home agent, the type is `struct ha_tunnel_data`, which contains information about the tunnel as was detailed in the description of `handle_aaa_reg_msg()`.

The SPI value stored in the binding is determined from the argument `mn_spi`, to `create_binding()`:

```
binding->spi = mn_spi->spi;
```

The IP address information maintained in the binding is set as follows:

```
inet_aton((char *) "192.168.242.2", &binding->mn_addr);

binding->ha_addr.s_addr = ext->req->ha_addr.s_addr;

binding->lower_addr.s_addr = ext->req->co_addr.s_addr;
```

It is important to note that in the case of the project implementation, the home agent address is determined from the registration request. This is shown by the assignment to `binding->ha_addr.s_addr` in the code. Ideally, with AAA, this should not be necessary. For the purposes of the project implementation, this simplified routing the registration request. Nevertheless, it is reasonable that the mobile node would know the address of its home agent.

The `id` of a binding is determined from the registration request as follows:

```
memcpy(&binding->id, ext->req->id, REG_REQ_ID_LEN);
```

The time of the binding's creation is also recorded:

```
binding->create_time = time(NULL);
```

The actual creation of a tunnel to the mobile node is done with the following call to `tunnel_add()`:

```
if (ok && create_tunnels &&
    tunnel_add(tunnels, binding->lower_addr, binding->tun_dev,
```

```
                ext->req->ha_addr, 1, TUNNEL_IPIP, 0) == NULL) {
            printf("Could not add tunnel to FA %s (COA=%s) for "
                 "MN %s\n", inet_ntoa(binding->lower_addr), co_addrstr,
                 mn_addrstr);
            ok = FALSE;
    }
```

The destination address of the tunnel is `binding->lower_addr`, the mobile node's care-of address. The local address is the address of the home agent, `ext->req->ha_addr`. The tunnel created is added to `tunnels`, which maintains information on all the tunnels that are in use by Dynamics. In this project, only IP-in-IP encapsulation (shown in the code as `TUNNEL_IPIP`) is used. This is the encapsulation mode that is required by the Mobile IP standard.

If the tunnel is successfully created, the following call to `dyn_ip_route_get()`, a function defined in `dyn_ip.c` is made:

```
    if (ok && create_tunnels &&
        dyn_ip_route_get(binding->mn_addr, t_data->arp_if, IFNAMSIZ)
        != 0) {
            printf("Could not get arp interface for MN %s\n",
                 mn_addrstr);
            ok = FALSE;
    }
```

The point of this call is to determine the correct interface for the mobile node's home address. The correct interface is stored in `t_data->arp_if`. As will be shown later, this is done to use proxy and gratuitous ARP as specified in the Mobile IP standard.

Upon the success of this, a call to `dyn_ip_route_replace()` is then made as was done with the AAA prototype:

```
    if (ok && create_tunnels &&
        dyn_ip_route_replace(binding->mn_addr, binding->tun_dev) != 0) {
            printf("Could not add route to MN %s\n",
                 mn_addrstr);
            ok = FALSE;
    }
```

The purpose of this is to update the routing table of the home agent so that all IP traffic destined to mobile node's home address (as given by `binding->mn_addr`) is directed to the tunnel device, `binding->tun_dev`.

If any of the tunnel creation or route updating steps fail as shown by setting `ok` to `FALSE` in the preceding code snippets, the memory allocated for the binding is de-allocated with `create_binding()` returning null as shown:

```
    if (!ok) {
            free(binding->data);
            free(binding);
            return NULL;
    }
```

On the other hand, if the tunnel creation and route updating steps are successful, the next step is to fulfill the Proxy and Gratuitous ARP requirements of Mobile IP. According to RFC 2002, these mechanisms are used to "attract and intercept any datagrams destined to the home

address" of any of a given home agent's registered mobile nodes. This is done with the following code:

```
if (create_tunnels) {
        ...
        proxyarp_add_item(binding->mn_addr, t_data->arp_if);

        proxyarp_gratuitous(binding->mn_addr, t_data->arp_if);
}
```

Using Proxy ARP, the home agent sends ARP replies on behalf of the mobile node when it is in a foreign network. This way the nodes making the ARP requests associate the home agent's link layer address with the mobile node's IP address. The call to `proxyarp_add_item()` in the preceding code snippet adds the mobile node's IP address to the cache of addresses on whose behalf the home agent sends ARP replies. With Gratuitous ARP, the home agent sends an unsolicited ARP packet to cause nodes that receive the packet to update their own ARP caches so that they associate the mobile node's IP address with the home agent's link layer address. It is to this end that the call to `proxyarp_gratuitous()` in the code is made.

With all the preceding steps completed, the number of bindings maintained by the home agent is incremented. Finally, a pointer to the binding created is returned. The following code shows this:

```
*bcount++;
return binding;
```

Chapter 4

Results

In this chapter, the results of the implementation are presented. These results are presented in the form of program output.

The mobile node begins by sending a registration request while in a foreign network as shown:

```
Registration Request
        type 1, opts 22, lifetime 300
        home_addr 0.0.0.0, ha_addr 192.168.242.1
        co_addr 192.168.240.2, id c71c93eb, 233777e0
        mn_nai: type 131, length 14, nai: mn@example.com
        mn_keyreq: type 134, length 12, vendor_id 5202, sub_type 5,
                spi 0, key len 0
        encaps_delivery
        MN_FA_KEY_REQ_AAA: type 40, subtype 7, length 4, MN SPI 1000 (0x000003e8), keylen 0
        MN_HA_KEY_REQ_AAA: type 42, subtype 7, length 4, MN SPI 1000 (0x000003e8), keylen 0
        gen_auth(MN-AAA): type 36, subtype 1, length 24, spi 12345, auth len 20
sending registration request to 192.168.242.1:434, type=1
```

The registration request shown here is sent directly to the home agent. It can be seen that the mobile node specifies the address of its home agent but not its home address. It uses the NAI of `mn@example.com`. The request also contains AAA key request extensions.

The home agent upon receiving the registration request, generates the keys requested for as shown:

```
auth_generate_key: alg=4, secret=aaa-test, keymat_len=8, nodeaddrlen=14
Adding MN-HA auth ext: alg=4 mn_spi.shared_secret=ac21d40... mn_spi.shared_secret_len=16
auth_generate_key: alg=4, secret=aaa-test, keymat_len=8, nodeaddrlen=14
Adding MN-FA auth ext: alg=5 fa_secret=3f5f21... fa_secret_len=16
```

The home agent then proceeds to make a binding for the mobile node and create a tunnel directly to it. This is shown in the output that follows:

```
Creating a new binding for MN
tunnel_add 192.168.240.2, type=1, key=0
        dyn_ip_tunnel_add(TUNL0,192.168.240.2,192.168.242.1)
tunnel_add => tun_dev=[TUNL0]
dyn_ip_route_get: 192.168.242.2 => eth0
proxyarp_add_item: ether - name=eth0, HW addr=00:02:b3:1f:32:50
proxyarp_add_item: added proxyarp for 192.168.242.2 to device eth0
proxyarp_gratuitous: sending gratuitous ARP - IPaddr: 192.168.242.2, HWaddr: ..., family=1
```

The tunnel is created as shown in the lines containing `tunnel_add` and `dyn_ip_tunnel_add`. The kernel routing table of the host on which the home agent is running is updated to reflect the creation of the tunnel. In addition, the ARP related mechanisms discussed in the previous chapter are also carried out.

Once all this is completed, the home agent sends the following registration reply to the mobile node:

```
Registration Reply
        type 3, code 0, lifetime 300
        home_addr 192.168.242.2, ha_addr 192.168.242.1
        id c71c93eb, 233777e0
        mn_nai: type 131, length 14, nai: mn@example.com
        MN_HA_KEY_MATERIAL_AAA: type 43, subtype 1, length 24, lifetime 600, keylen 20
                aaa_spi=12345, ha_spi=12346, alg_id=2, replay_method=2
        mh_auth: type 32, length 20, spi 1000, auth len 16
        MN_FA_KEY_MATERIAL_AAA: type 41, subtype 7, length 22, keylen 22
                lifetime=600, aaa_spi=12345, fa_spi=100000, alg_id=3
        challenge: type 132, length 4, challenge: 62A9AA7B
        mf_auth: type 33, length 24, spi 1000, auth len 20
```

One item of particular interest is the provision of a home address for the mobile node. The reply also contains key material generated from the message received from the mobile node.

The mobile node receives the registration reply and now has the keys it needs for Mobile IP operation:

```
MN-FA key: 3f5f21d5bc8e204153ee55933a9f2973
MN-HA key: ac21d40f5066194009c11882263a8d22
add_fa_spi: adding dynamic security association (SPI=100000, addr=192.168.242.1)
```

The mobile node also determines its home address from the registration reply as shown:

```
Discovered MN home address from reply: 192.168.242.2
Registration accepted
```

Finally, the mobile node sets up a tunnel to the home agent as shown:

```
Start tunneling - FA addr 192.168.242.1
Adding tunnel TUNLMNA => 192.168.242.1
Setting default route to TUNLMNA
```

The output above also shows that the mobile node sets this tunnel as the default route.

At the mobile node, the creation of the tunnel can be verified with the Linux `ifconfig` command which gives:

```
TUNLMNA   Link encap:IPIP Tunnel  HWaddr
          inet addr:192.168.242.2  P-t-P:192.168.242.2  Mask:255.255.255.255
```

The updating of the routing table, on the other hand, is verifed with the Linux command, `route -n` to give:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 TUNLMNA
```

The tunnel, shown as `TUNLMNA`, becomes the default route.

Similarly, the creation of the tunnel at the home agent is verified with the following output:

```
TUNL0     Link encap:IPIP Tunnel  HWaddr
          inet addr:192.168.242.1  P-t-P:192.168.242.1  Mask:255.255.255.255
...
```

The updated routing table at the home agent is as follows:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.242.2   0.0.0.0         255.255.255.255 UH    0      0        0 TUNL0
...
```

Chapter 5

Further Work

There remains considerable work that could be done beyond the implementation described in this report. In this project's implementation, for instance, tunnels can only be made using colocated care-of addresses. Future work could extend this so that foreign agent care-of addresses may also be used.

In addition, because this project describes a prototype, future work could involve a more comprehensive implementation that made use of AAA servers as described in RFC 2977. This could make use of RADIUS servers such as FreeRADIUS, an open-source implementation of RADIUS. The Diameter protocol is expected to replace RADIUS as the AAA protocol of choice particularly with developments in mobile computing. Future work could therefore incorporate the use of Diameter for AAA services.

Bibliography

[1] K. Ahmavaara, H. Haverinen, and R. Pichna. Interworking architecture between 3GPP and WLAN systems. *Communications Magazine, IEEE*, 41(11):74–81, 2003.

[2] N. Banerjee, Wei Wu, and S.K. Das. Mobility support in wireless Internet. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 10(5):54–61, 2003.

[3] M.M. Buddhikot, G. Chandranmenon, Seungjae Han, Yui-Wah Lee, S. Miller, and L. Salgarelli. Design and implementation of a WLAN/cdma2000 interworking architecture. *Communications Magazine, IEEE*, 41(11):90–100, 2003.

[4] P. Calhoun. RFC 2794: Mobile IP network access identifier for IPv4, March 2000.

[5] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. RFC 3588: Diameter base protocol, September 2003.

[6] M. Cappiello, A. Floris, and L. Veltri. Mobility amongst heterogeneous networks with AAA support. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 4, pages 2064–2069 vol.4, 2002.

[7] Thaison Dao, H.A. Chan, and J. Rejeb. Interworking between wireless LAN and CDMA2000. In *Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE*, volume 2, pages 1215–1220 Vol.2, 2003.

[8] J. De Vriendt, P. Laine, C. Lerouge, and Xiaofeng Xu. Mobile network evolution: a revolution on the move. *Communications Magazine, IEEE*, 40(4):104–111, 2002.

[9] Ashutosh Dutta, Tao Zhang, Sunil Madhani, Kenichi Taniuchi, Kensaku Fujimoto, Yasuhiro Katsube, Yoshihiro Ohba, and Henning Schulzrinne. Secure universal mobility for wireless internet . In *Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 71–80. ACM Press, Philadelphia, PA, USA, 2004.

[10] W.M. Eddy. At what layer does mobility belong? *Communications Magazine, IEEE*, 42(10):155–159, 2004.

[11] S. Glass, T. Hiller, S.Jacobs, and C. Perkins. RFC 2977: Mobile IP authentication, authorization and accounting requirements, October 2000.

[12] E. Gustafsson and A. Jonsson. Always best connected. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 10(1):49–55, 2003.

[13] Jonathan Hassell. *RADIUS*. O'Reilly, 1st edition, 2003.

[14] T.R. Henderson. Host mobility for IP networks: a comparison. *Network, IEEE*, 17(6):18–26, 2003.

[15] Suk Yu Hui and Kai Hau Yeung. Challenges in the migration to 4G mobile systems. *Communications Magazine, IEEE*, 41(12):54–59, 2003.

[16] G.M. Koien and T. Haslestad. Security aspects of 3G-WLAN interworking. *Communications Magazine, IEEE*, 41(11):82–88, 2003.

[17] ByungGil Lee, HyunGon Kim, and KilHoum Park. *An AAA Application Protocol Design and Service for Secure Wireless Internet Gateway Roaming*. Lecture Notes in Computer Science. 2344 edition, 2002.

[18] A. Mahapatra and R. Uma. Authentication in an integrated 802.1X based WLAN and CDMA2000-1X network. In *Communications, 2003. APCC 2003. The 9th Asia-Pacific Conference on*, volume 1, pages 227–231 Vol.1, 2003.

[19] C. Metz. AAA protocols: authentication, authorization, and accounting for the Internet. *Internet Computing, IEEE*, 3(6):75–79, 1999.

[20] Ai-Chun Pang, Jyh-Cheng Chen, Yuan-Kai Chen, and P. Agrawal. Mobility and session management: UMTS vs. cdma2000. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(4):30–43, 2004.

[21] C. Perkins. RFC 2002: IP mobility support, October 1996.

[22] C. Perkins. RFC 3012: Mobile IPv4 challenge/response extensions, November 2000.

[23] C. Perkins. RFC 3957: Authentication, authorization, and accounting (AAA) registration keys for mobile IPv4, March 2005.

[24] C.E. Perkins. Mobile networking through Mobile IP. *Internet Computing, IEEE*, 2(1):58–69, 1998.

[25] C.E. Perkins. Mobile IP joins forces with AAA. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 7(4):59–61, 2000.

[26] C.E. Perkins. Mobile IP. *Communications Magazine, IEEE*, 40(5):66–82, 2002.

[27] C. Politis, K.A. Chew, N. Akhtar, M. Georgiades, R. Tafazolli, and T. Dagiuklas. Hybrid multilayer mobility management with AAA context transfer capabilities for all-IP networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(4):76–88, 2004.

[28] Neeli R. Prasad, Mahbubul Alam, and Marina Ruggieri. Light-Weight AAA Infrastructure for Mobility Support Across Heterogeneous Networks . *Wirel. Pers. Commun.*, 29(3-4):205–219, 2004.

[29] C. Rensing, M. Karsten, and B. Stiller. AAA: a survey and a policy-based architecture and framework. *Network, IEEE*, 16(6):22–27, 2002.

[30] C. Rigney, S. Willens, A. Rubens, and W. Simpson. RFC 2865: Remote authentication dial in user service (RADIUS), June 2000.

[31] L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller. Emerging authentication and key distribution in wireless IP networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 10(6):52–61, 2003.

[32] A.K. Salkintzis. Interworking techniques and architectures for WLAN/3G integration toward 4G mobile data networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(3):50–61, 2004.

[33] B. Sarikaya and S. Gurivireddy. Evaluation of CDMA2000 support for IP micromobility handover and paging protocols. *Communications Magazine, IEEE*, 40(5):146–149, 2002.

[34] Minghui Shi, Xuemin Shen, and J.W. Mark. IEEE 802.11 roaming and authentication in wireless LAN/cellular mobile networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(4):66–75, 2004.

[35] James D. Solomon. *Mobile IP, the Internet unplugged*. Prentice Hall series in computer networking and distributed systems. PTR Prentice Hall, Upper Saddle River, N.J., 1998.

[36] Yuh-Min Tseng, Chou-Chen Yang, and Jiann-Haur Su. An efficient authentication protocol for integrating WLAN and cellular networks. In *Advanced Communication Technology, 2004. The 6th International Conference on*, volume 1, pages 416–420, 2004.

[37] Wenye Wang, Wei Liang, and Avesh K. Agarwal. Integration of authentication and mobility management in third generation and WLAN data networks. *Wireless Communications and Mobile Computing*, 5(6):665–678, 2005.

[38] Chou Chen Yang, Min Shiang Hwang, Jian Wei Li, and Ting Yi Chang. *A Solution to Mobile IP Registration for AAA*. Lecture Notes in Computer Science. 2524 edition, 2003.